ＡＤＳＰ３２４-０３

４チャンネル 12bit A/D & D/A ボード
ソフトウェア・ユーザーズ・マニュアル
ＡＤＳＰ６７４－００用

中部電機株式会社

目次

1. 概要

ADSP324-03 サポートソフトウェアは、ADSP324-03 を使用する為の基本機能を含んだ BIOS プログラム(A03_67BIOS)及び、それを用いたサンプルプログラムから構成されています。

2. 機能一覧

A03_67BIOS には以下の機能があります。

1) ADSP324-03 ボードの初期化
2) ソフトウェア同期の A/D & D/A 変換機能
3) タイマー同期の A/D & D/A 変換機能
4) トリガー待機機能

3. 供給形態

A03_67bios はソースファイル及び、COFF ファイル形式のオブジェクト、ライブラリ形式で供給されています。オブジェクトリンク又はライブラリリンクのいずれかの方法で利用してください。ライブラリリンクにて利用する場合は、A03_67bios.h と A03_67bios.lib を C6x_C_DIR 環境変数の示すディレクトリにコピーしておけば簡単に利用することができます。

4. 供給ファイル一覧

Readme.txt	A03_67bios の簡単な説明が書かれています。
A03_67bios.c	A03_67bios のソースファイル
A03_67bios.h	A03_67bios を使用する為のヘッダファイル
A03_67bios.obj	A03_67bios のオブジェクトファイル
Timer.c	A03_67bios にて使用されているタイマ関連関数ソースファイル
Timer.h	Timer を使用する為のヘッダファイル
Timer.obj	Timer のオブジェクトファイル
A03_67bios.lib	A03_67bios のライブラリファイル
A03_67.cmd	A03_67bios を用いるためのコマンドファイル
Sample.c	A03_67bios を用いたサンプルプログラム

5. 実装方法

１）COFF ファイル形式のオブジェクトリンク
カレント・ディレクトリに A03_67bios.obj、Timer.obj をコピーして、ユーザ・プログラムとリンクしてください。

A03_67bios.c、Timer.c をプロジェクトに追加します。
「Project」-「Add Files to Project」
A03_67bios.c、Timer.c を選択 → 開く

２）ライブラリリンク
環境変数 C6x_C_DIR の示すディレクトリに、A03_67bios.h と A03_67bios.lib をコピーすることにより、リンカの -l オプションでライブラリを指定してリンクして下さい。

環境変数 C6x_C_DIR の内容を、確認します。
C6x_C_DIR = C:¥ti¥c6000¥cgtools¥include ； C:¥ti¥c6000¥cgtools¥lib...
通常は、DSP の C コンパイラのライブラリが格納されているパスがセットされています。
上記の場合は
A03_67bios.h を　C:¥ti¥c6000¥cgtools¥include
A03_67bios.lib を C:¥ti¥c6000¥cgtools¥lib
へコピーします。
プログラムのリンク時に A03_67bios.lib をリンクして下さい。
なお、C6x_C_DIR 環境変数の設定については、TI の C コンパイラの取り扱い説明書を参照して下さい。

5. 関数一覧

○ 初期化関数

A67X_03init	ボードの初期化及びライブラリの初期化を行います

○ 正規化関数

A67X_03adnorm	A/D データを浮動小数点形式に変換します
A67X_03danorm	浮動小数点形式データを D/A データに変換します

○ A/D 変換関数

A67X_03adinput	指定 A/D チャンネルの A/D 変換を行います
A67X_03adperiod1	指定 A/D チャンネルの A/D 変換を行います（定周期）
A67X_03adperiod2	指定 A/D チャンネルの A/D 変換を行います（外部周期）

○ D/A 変換関数

A67X_03daoutput	指定 D/A チャンネルの D/A 変換を行います
A67X_03daperiod1	指定 D/A チャンネルの D/A 変換を行います（定周期）

○ トリガー関数

A67X_03trigger	トリガー入力を待ちます

6. 関数詳細

関数名　　ボードの初期化及びライブラリの初期化

記述　　int　　A67X_03init(max, adrs, param);

引数　　int　　max;　　// 03 ボードの実装枚数
A03_67BD_PORT　　mode;　　// 03 ボードのベースアドレス
A03_67PARAM　　base;　　// 03 ボードの初期化パラメータ

戻り値　　_ERR　　パラメータ異常
_NER　　正常に初期化されました

説明　　ADSP32X_03 の初期化(D/A の出力を 0[V]に設定)します。また、ライブラリの諸設定を行います。
ボード実装枚数の指定は 1～4 が指定可能です。
ボードのベースアドレスは、1 枚目のボードから 10h ステップで各ボードを設定し、1 枚目のベースアドレスを指定してください。
初期化構造体の説明は、第 7 章・構造体の説明を参照して下さい。

使用例

```
#include        <A03_67bios.h>

#define         BD_BASE         0x3000000
#define         BD_MAX          4

A03_67BD_PORT   *Port = (A03_67BD_PORT *)BD_BASE;
A03_67PARAM     Param[BD_MAX] =
{
    {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}},   // BD1 のパラメータ
    {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}},   // BD2 のパラメータ
    {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}},   // BD3 のパラメータ
    {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}}    // BD4 のパラメータ
};

void  main(void)
{
    A67X_06init(BD_MAX, Prot, &Param);
}
```

関数名　　A/D データの正規化

記述　　int　　A67X_03adnorm(top, chc, dtc, src, dst);

引数
```
int             top;            // 先頭チャンネル番号
int             chc;            // 変換チャンネル数
int             dtc;            // 変換データ数
int             *src;           // A/D 変換データ格納ポインタ
float           *dst;           // 正規化データ格納ポインタ
```

戻り値
```
_ERR    異常終了
_NER    正常終了
```

説明　　A/D 変換されたデータを、A/D の入力レンジと入力ゲインで設定されたデータを基に浮動小数点形式の値に変換します。
入力レンジと入力ゲインは初期化時の値が使用されます。
データの配列を以下に示します。データ配列については第 8 章・データ配列を参照して下さい。
```
    src + 0             : 先頭チャンネルのデータ
    src + chc - 1       : 最終チャンネルのデータ
    src + chc           : 次のデータ
```

使用例
```
#include        <A03_67bios.h>

int             i;
int             AD_BUFF[4];
float           AD_DATA[4];

void  main(void)
{
    for( ;; ){
        A67X_03adinput(0, 1, AD_BUFF);
        A67X_03adnorm(0, 1, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 4; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

関数名　　D/A データの正規化

記述　　int　　A67X_03danorm(top, chc, dtc, src, dst);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　dtc;　　// 変換データ数
float　　*src;　　// D/A 変換データ格納ポインタ
int　　*dst;　　// 正規化データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　D/A 変換する浮動小数点形式のデータを、D/A の出力レンジで設定されたデータを基に D/A 変換器の出力形式に変換します。
出力レンジは初期化時の値が使用されます。
データの配列を以下に示します。
src + 0　　：先頭チャンネルのデータ
src + chc - 1　　：最終チャンネルのデータ
src + chc　　：次のデータ

使用例

```
#include        <A03_67bios.h>

int             i, j;
int             DA_BUFF[256];
float           DA_DATA[256];

void  main(void)
{
    for( ;; ){
        for( j = 0; ; j = (j + 10) % 360 ){
            DA_DATA[0] = sin(PI2 * (float)j / 360.0) * 5.0;
            A67X_03danorm(0, 1, 1, DA_DATA, DA_BUFF);
            A67X_03daoutput(0, 1, DA_BUFF);
        }
    }
}
```

関数名　　指定チャンネルの A/D 変換（ソフトウェア同期）

記述　　int　　A67X_03adinput(top, chc, buf);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　*buf;　　// A/D 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたチャンネル範囲を A/D 変換します。
データの配列を以下に示します。
src + 0　　：先頭チャンネルのデータ
src + chc - 1　　：最終チャンネルのデータ
src + chc　　：次のデータ

使用例

```
#include        <A03_67bios.h>

int             i;
int             AD_BUFF[4];
float           AD_DATA[4];

void  main(void)
{
    for( ;; ){
        A67X_03adinput(0, 1, AD_BUFF);
        A67X_03adnorm(0, 1, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 4; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
```

関数名　　指定チャンネルの D/A 変換（ソフトウェア同期）

記述　　int　　A67X_03daoutput(top, chc, buf);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　*buf;　　// D/A 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたチャンネル範囲に D/A 変換します。
データの配列を以下に示します。
src + 0　　：先頭チャンネルのデータ
src + chc - 1　　：最終チャンネルのデータ
src + chc　　：次のデータ

使用例

```
#include        <A03_67bios.h>

int             i;
int             DA_BUFF[256];
float           DA_DATA[256];

void  main(void)
{
    for( ;; ){
        for( i = 0; ; i = (i + 10) % 360 ){
            DA_DATA[0] = sin(PI2 * (float)i / 360.0) * 5.0;
            A67X_03danorm(0, 1, 1, DA_DATA, DA_BUFF);
            A67X_03daoutput(0, 1, DA_BUFF);
        }
    }
}
```

関数名　　指定チャンネルの A/D 変換（タイマー同期）

記述　　int　　A67X_03adperiod1(top, chc, dtc, prod, buf);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　dtc;　　// 変換データ数
int　　prod;　　// 変換周期
int　　*buf;　　// A/D 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたチャンネル範囲を A/D 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
データの配列を以下に示します。
src + 0　　：先頭チャンネルのデータ
src + chc - 1　　：最終チャンネルのデータ
src + chc　　：次のデータ

使用例

```
#include        <A03_67bios.h>

int             i;
int             AD_BUFF[4];
float           AD_DATA[4];

void   main(void)
{
    asm("       mvc     CSR, b0    ");  // Get CRS
    asm("       or      1, b0, b0  ");  // Get ready to GIE
    asm("       mvc     b0, CSR    ");  // Set GIE

    A67X_03init(1, Port, &Param);

    for( ;; ){
        A67X_03adperiod1(0, 4, 1, 100, AD_BUFF);
        A67X_03adnorm(0, 4, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 4; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

関数名　　指定チャンネルの D/A 変換（タイマー同期）

記述　　int　　A67X_03daperiod1(top, chc, dtc, prod, buf);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　dtc;　　// 変換データ数
int　　prod;　　// 変換周期
int　　*buf;　　// D/A 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたチャンネル範囲に D/A 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
データの配列を以下に示します。
src + 0　　：先頭チャンネルのデータ
src + chc - 1　　：最終チャンネルのデータ
src + chc　　：次のデータ

使用例

```
#include        <A03_67bios.h>

int             DA_BUFF[4*256];

void  main(void)
{
    int         i;
    float       a;

    asm("       mvc     CSR, b0    ");  // Get CRS
    asm("       or      1, b0, b0  ");  // Get ready to GIE
    asm("       mvc     b0, CSR    ");  // Set GIE

    A67X_03init(1, Port, &Param);

    a = 3.141592653 * 2 / 64.0;
    for( i = 0; i < 256; i++ ){
        DA_DATA[i] = 5 * sin(a * i);
    }
    for( ;; ){
        A67X_03danorm(0, 1, 256, DA_DATA, DA_BUFF);
        A67X_03daperiod1(0, 1, 256, 100, DA_BUFF);
    }
}
```

関数名　　トリガー待ち

記述　　int　　A67X_03trigger(bd, lvl, slope);

引数　　int　　bd;　　// ボード番号
double　　lvl;　　// トリガーレベル
int　　slope;　　// トリガースロープ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたボードでトリガー監視をします。
レベルの指定は、浮動小数点形式で、±10[V]の指定が可能です。
スロープの指定は、0 なら立ち上がりスロープ、0 以外なら立下りスロープです。
トリガーを検出するまで戻りません。

使用例

```
#include        <A03_67bios.h>

void  main(void)
{
    int         i;

    asm("       mvc     CSR, b0    ");  // Get CRS
    asm("       or      1, b0, b0  ");  // Get ready to GIE
    asm("       mvc     b0, CSR    ");  // Set GIE

    A67X_03init(1, Port, &Param);

    a = 3.141592653 * 2 / 64.0;
    for( i = 0; i < 256; i++ ){
        DA_DATA[i] = 5 * sin(a * i);
    }
    for( ;; ){
        A67X_03trigger(0, 2.5, 0);
    }
}
```

注意

この関数を使用する場合は DSW104-1, 2, 3 を ON, OFF, OFF に設定して下さい。

関数名　　指定チャンネルの A/D 変換（外部同期）

記述　　int　　A67X_03adperiod2(top, chc, dtc, prod, buf);

引数　　int　　top;　　// 先頭チャンネル番号
int　　chc;　　// 変換チャンネル数
int　　dtc;　　// 変換データ数
int　　prod;　　// 変換周期
int　　*buf;　　// A/D 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定されたチャンネル範囲を A/D 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
通常は、TCLK0 の設定も行うので、ディップスイッチの設定で変換クロックの選択が TCLK0 なら、指定した変換周期で変換されます。外部クロックを選択した場合は、外部クロック入力端子に特定のクロックを入力しておいて下さい。
データの配列を下記に示します。
src + 0　　: 先頭チャンネルのデータ
src + chc - 1　　: 最終チャンネルのデータ
src + chc　　: 次のデータ(変換周期毎)

使用例

```
#include        <A03_67bios.h>

int             i;
int             AD_BUFF[4*256];

void  main(void)
{
    for( ;; ){
        A67X_03adperiod2(0, 4, 1, 100, AD_BUFF);
        A67X_03adnorm(0, 4, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 4; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

注意

この関数を使用する場合は DSW104-1, 2, 3 を OFF, OFF, ON に設定して下さい。

7. 構造体の説明

構造体は、typedef を用いて<AO3_67bios.h>の中で定義されています。

1) A/D & D/A ポートの定義

```
typedef  struct
{
     unsigned  int       AD[4],          // A/D 入力ポート
                         DA[4],          // D/A 出力ポート
                         AD_BUSY,        // 変換中ポート
                         CTRL,           // 制御ポート
                         TRIG_LVL,       // トリガーレベルポート
                         AD_GAIN,        // A/D 入力ゲインポート
                         INT_RST,        // 割り込みリセットポート
                         DMAD[3];        // 空き
}AO3_67BD_PORT;
```

2) 初期化構造体の定義

```
typedef  struct
{
    unsigned  int        AD_Gain[4],     // A/D 入力ゲイン
                                         (0: 1 倍, 1: 2 倍, 2: 4 倍, 3: 8 倍)
                         AD_range[4],    // A/D 入力レンジ
                                         (0: ±10[V], 1: ±5[V])
                         DA_range[4],    // D/A 出力レンジ
                                         (0: ±10V, 1: ±5V, 2: ±2.5V, 3: +10V, 4: +5V)
}AO3_67PARAM;
```

8. データ配列

chc を 4 にした場合。

	データ 1	データ 2	
Ch0	src+0	src+chc	
Ch1	src+1	src+chc+1	
Ch2	src+2	src+chc+2	
Ch3	src+chc-1	src+chc+3	

．．．

9. ボード制御ソフトを書く上での注意

ボード制御ソフトをユーザーサイドで独自に作る場合においての注意点を説明します。

1) ベクタの使用

割り込みを複数のボードで使用する上で、ベクタ番号は重要な役割を持ちます。ベクタ番号の設定は、DSW104 で行うことができボード間で重複しないように設定します。

例)
1 枚目のボード DSW104-1 を ON、他は OFF
2 枚目のボード DSW104-2 を ON、他は OFF

このように設定しておくことにより、どのボードから割り込み要求が来たかを知ることができるようになります。

方法は、ベースアドレスの下位 20 ビットが 3fffc のアドレス(nnn3fffc)番地を読むことによって行います。()内の nnn は、ボードのベースアドレスの上位 12 ビットの設定です。
この事からわかる様に、割り込みを使用するボード全てのベースアドレスの上位 12 ビットは同一の設定である必要があります。

ベクトポートは割り込みが発生していない場合、下位 8 ビット全てが 1 です。割り込みが発生した場合、割り込み要求を出しているボードの DSW104 の ON 位置のビットが 0 になります。

以下に、ベクタを用いたプログラムを示します。

例) ボードが 2 枚実装されているものとし、1 枚目はベクタ番号 1(DSW104-1 を ON)、2 枚目はベクタ番号 2(DSW104-2 を ON)とします。

```
#define     BD_MAX          2                                   // 実装ボード枚数
#define     BD_BASE         0x3000000                           // ボードのベースアドレス
#define     VECT_PORT(a)    (unsigned int *)(((unsigned int)(a) & 0xff00000) + 0x3fffc)

static      A06_67BD_PORT   *FST_BASE;                          // 1 枚目のボードアドレス
static      A06_67BD_PORT   *BD_BASE[BD_MAX];                   // 各ボードのベースアドレス
static int  VECT_MASK = 0;                                      // ベクタマスク
```

```
int         *int_bd = (int*)0x303fffc;                    // 割り込みボード確認用

interrupt void BD1(void)
{
    printf("割り込みプログラム¥n");
}

interrupt void BD2(void)
{
    printf("割り込みプログラム¥n");
}

//=============================================
//          割り込み処理
//=============================================
interrupt void c_int90(void)
{
    int             int_no;
    unsigned int    vect, bd, bit;

    asm("           nop     2               ");     // 無効割り込み検査
    asm("           mvc     IFR, b3         ");
    asm("           mvk     0080h, b4       ");
    asm("           and     b4, b3, b0      ");
    asm("   [b0]    b       int_exit        ");
    asm("           nop     5               ");

    vect = ~(*VECT_PORT(FST_BASE) | (~VECT_MASK));   // 割り込み発生状況
    for( bd = 0, bit = 1; bd < BD_MAX; ++bd, bit <<= 1 ){// 各ボードの割り込みスキャン
        if( vect & bit ){                            // 有効割り込み確認
            if( int_bd & 0x0f == 0x0e )
                int_no = 1;
            else if( int_bd & 0x0f == 0x0d )
                int_no = 2;

            BD_BASE[bd]->INTR = 0;                   // 割り込みのリセット
            if( int_no == 1 )
                BD1();                               // 1枚目のボードの処理
            else if( int_no == 2 )
                BD2();                               // 2枚目のボードの処理
            }
    }
    asm(" int_exit:                         ");
}
```

2) 電源投入直後の状態

ADSP324-03 ボードの電源投入直後の状態は、D/A の全てのチャンネルの出力（トリガーレベル設定用 D/A も含む）が設定レンジの最大値になっています。このため、電源投入直後からボードを初期化するまでの間に、サーボ系で D/A を使用する場合は、インターロックをとるなどして誤動作を回避してください。

3) ユーザソフトをアセンブラで記述する場合

拡張バスのメモリにデータを書き込む場合は “STB”“STH”命令は使用しないで下さい。以下に説明を示します。

STB 命令では 8 bit 単位で、STH 命令では 16 bit 単位でメモリの読み書きを行います。
しかし、拡張ボードにデータを書き込む場合は 32 bit(1 word)単位で実効する必要があります。

以上の様に、それぞれ扱うデータサイズが異なるため STB・STH 命令を使用した場合は不完全なデータになります。

4) 割り込み使用時

割り込みを使用する関数を実行するときは、ユーザプログラムにてグローバル割り込みレジスタ(GIE)を有効（Enable）設定してください。

例）

```
// GIE enable
    asm("       mvc         CSR, b0         ");
    asm("       or          1, b0, b0       ");
    asm("       mvc         b0, CSR         “);
```

・本マニュアルの内容は製品の改良のため予告無しに
変更される事がありますので、ご了承下さい。

中部電機株式会社

〒440-0004　愛知県豊橋市忠興３丁目２－８

TEL <0532>61-9566

FAX <0532>63-1081

URL　: http://www.chubu-el.co.jp

E-mail : csg@chubu-el.co.jp

2002. 11　第３版発行